https://mp.weixin.qq.com/s/qOMO0LIdA47j3RjhbCWUEQ

导语

Http 缓存机制作为 web 性能优化的重要手段，对从事 Web 开发的小伙伴们来说是必须要掌握的知识，但最近我遇到了几个缓存头设置相关的题目，发现有好几道题答错了，有的甚至在知道了正确答案后依然不明白其原因，可谓相当的郁闷呢！！为了确认下是否只是自己理解不深，我特意请教了其他几位小伙伴，发现情况也或多或少和我类似。

为了不给大家卖关子，下面我贴出2道题，大家可以尝试解答下：

以下为 **page.html** 内容：

```
<!DOCTYPE html><html xmlns="http://www.w3.org/1999/xhtml"><head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>page页</title></head><body>
    <img src="images/head.png" />
    <a href="page.html">重新访问page页</a></body></html>
```

首次访问该页面，页面中 **head.png** 响应头信息如下：

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Content-Type: image/png
Last-Modified: Tue, 08 Nov 2016 06:59:00 GMT
Accept-Ranges: bytes
Date: Thu, 10 Nov 2016 02:48:50 GMT
Content-Length: 3534
```

- **问题1**：请问当点击“重新访问 page 页”链接重新加载该页面后， head.png 如何二次加载？
- **问题2**：如果将上述信息中的 Cache-Control 设置为 private，那么结果又会如何呢？

以上2道题，如果你能全部答对（哈哈，还请仔细确认下 why，以防歪打正着），那么恭喜你，你已对这些知识理解非常透彻了，我后面讲的内容你可以忽略，否则还请继续陪我往下唠唠吧！

首先回到开篇提到很多小伙伴（包括我）在解答 Http 缓存题目时栽跟头的问题，我觉得出现这种现象的根本原因在于我们吸收的知识还不够体系化，平时我们在学习这些知识时多半将其当作知识点来记，什么这个缓存头作什么、那个缓存头作什么用的，但实际中缓存头往往是多个之间相互配合协同工作的，有一套完整的工作体系。

今天我将按自己的理解，从系统体系化角度来讲讲 Http 缓存头是如何协同工作的（不正确的地方还请指正，但请不要喷我哦）：

HTTP 缓存体系

首先我将 Http 缓存体系分为以下三个部分：

1. 缓存存储策略

用来确定 Http 响应内容是否可以被客户端缓存，以及可以被哪些客户端缓存

这个策略的作用只有一个，用于决定 Http 响应内容是否可缓存到客户端

对于 Cache-Control 头里的 Public、Private、no-cache、max-age 、no-store 他们都是用来指明响应内容是否可以被客户端存储的，其中前4个都会缓存文件数据（关于 no-cache 应理解为“不建议使用本地缓存”，其仍然会缓存数据到本地），后者 no-store 则不会在客户端缓存任何响应数据。另关于 no-cache 和 max-age 有点特别，我认为它是一种混合体，下面我会讲到。

通过 Cache-Control：Public 设置我z+们可以将 Http 响应数据存储到本地，但此时并不意味着后续浏览器会直接从缓存中读取数据并使用，为啥？因为它无法确定本地缓存的数据是否可用（可能已经失效），还必须借助一套鉴别机制来确认才行， 这就是我们下面要讲到的“缓存过期策略”。

2. 缓存过期策略

客户端用来确认存储在本地的缓存数据是否已过期，进而决定是否要发请求到服务端获取数据

这个策略的作用也只有一个，那就是决定客户端是否可直接从本地缓存数据中加载数据并展示（否则就发请求到服务端获取）

刚上面我们已经阐述了数据缓存到了本地后还需要经过判断才能使用，那么浏览器通过什么条件来判断呢？ 答案是：Expires，Expires 指名了缓存数据有效的绝对时间，告诉客户端到了这个时间点（比照客户端时间点）后本地缓存就作废了，在这个时间点内客户端可以认为缓存数据有效，可直接从缓存中加载展示。

不过 Http 缓存头设计并没有想象的那么规矩，像上面提到的 Cache-Control（这个头是在Http1.1里加进来的）头里的 no-cache 和 max-age 就是特例，它们既包含缓存存储策略也包含缓存过期策略，以 max-age 为例，他实际上相当于：

```
Cache-Control：public/private（这里不太确定具体哪个）
Expires：当前客户端时间 + maxAge 。
```

而 Cache-Control：no-cache 和 Cache-Control：max-age=0 （单位是秒）相当

这里需要注意的是：

1. Cache-Control 中指定的缓存过期策略优先级高于 Expires，当它们同时存在的时候，后者会被覆盖掉。
2. 缓存数据标记为已过期只是告诉客户端不能再直接从本地读取缓存了，需要再发一次请求到服务器去确认，并不等同于本地缓存数据从此就没用了，有些情况下即使过期了还是会被再次用到，具体下面会讲到。

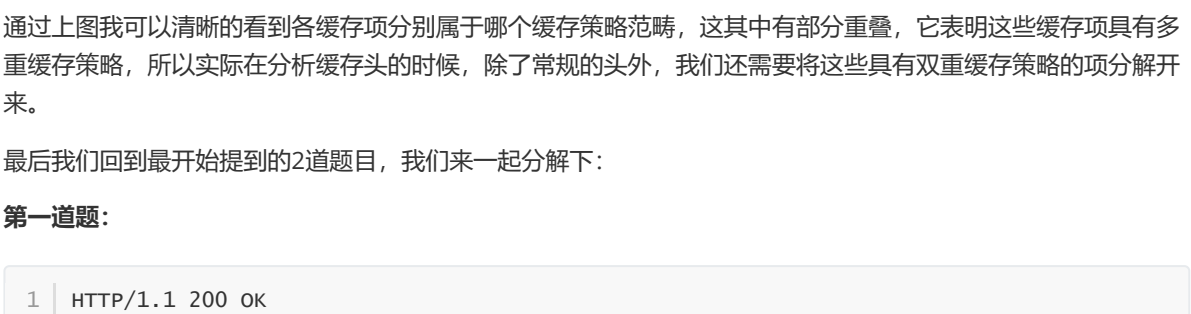
3. 缓存对比策略

将缓存在客户端的数据标识发往服务端，服务端通过标识来判断客户端 缓存数据是否仍有效，进而决定是否要重发数据。

客户端检测到数据过期或浏览器刷新后，往往会重新发起一个 http 请求到服务器，服务器此时并不急于返回数据，而是看请求头有没有带标识（ If-Modified-Since、If-None-Match）过来，如果判断标识仍然有效，则返回304告诉客户端取本地缓存数据来用即可（这里要注意的是你必须要在首次响应时输出相应的头信息（Last-Modified、ETags）到客户端）。至此我们就明白了上面所说的本地缓存数据即使被认为过期，并不等于数据从此就没用了的道理了。

关于 Last-Modified，这个响应头使用要注意，可能会影响到缓存过期策略，具体原因，后面我会通过解答开篇提到的2道题来作说明。

以上就是我所认识的缓存策略，下面我将缓存策略三要素和常用的几个缓存头（项）结合一起，让大家更清晰的认识到它们之间的关系：



通过上图我可以清晰的看到各缓存项分别属于哪个缓存策略范畴，这其中有部分重叠，它表明这些缓存项具有多重缓存策略，所以实际在分析缓存头的时候，除了常规的头外，我们还需要将这些具有双重缓存策略的项分解开来。

最后我们回到最开始提到的2道题目，我们来一起分解下：

第一道题：

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Content-Type: image/png
Last-Modified: Tue, 08 Nov 2016 06:59:00 GMT
Accept-Ranges: bytes
Date: Thu, 10 Nov 2016 02:48:50 GMT
Content-Length: 3534
```

分析上述 Http 响应头发现有以下两项与缓存相关：

```
Cache-Control: no-cache
Last-Modified: Tue, 08 Nov 2016 06:59:00 GMT
```

我们上面讲到了 Cache-Control: no-cache 相当于 Cache-Control: max-age=0，且他们都是多重策略头，我们需将其分解：

Cache-Control: no-cache 等于 Cache-Control: max-age=0，
接着 Cache-Control: max-age=0 又可分解成：

```
Cache-Control: public/private （不确定是二者中的哪一个）
Expires: 当前时间
```

最终我们得到了以下完整的缓存策略三要素：

缓存策略类型	缓存策略值	结果	备注
缓存存储策略	cache-control: public/private	响应数据会被缓存到客户端	由 cache-control: no-cache 拆解而来
缓存过期策略	Expires: 当前时间	立马过期，二次资源访问浏览器会重新发起Http请求	
缓存对比策略	Last-Modified: Tue, 08 Nov 2016 06:59:00 GMT	浏览器会携带该值去服务端比对，比对成功则返回304，服务端提示浏览器从本地加载数据，否则返回200并响应数据。	

所以最终结果是：浏览器会再次请求服务端，并携带上 Last-Modified 指定的时间去服务器对比：

- a）**对比失败**：服务器返回200并重发数据，客户端接收到数据后展示，并刷新本地缓存。
- b）**对比成功**：服务器返回304且不重发数据，客户端收到304状态码后从本地读取缓存数据。以下为模拟此种情况下请求后的抓包情况：

这道题本身不难，但若认为 no-cache 不会缓存数据到本地，那么你理解起来就会很矛盾，因为如果文件数据没有被本地缓存，服务器返回304后将会无法展示出图片内容，但实际上它是能正常展示的。这道题很好的证明了 no-cache 也会缓存数据到本地这一说法。

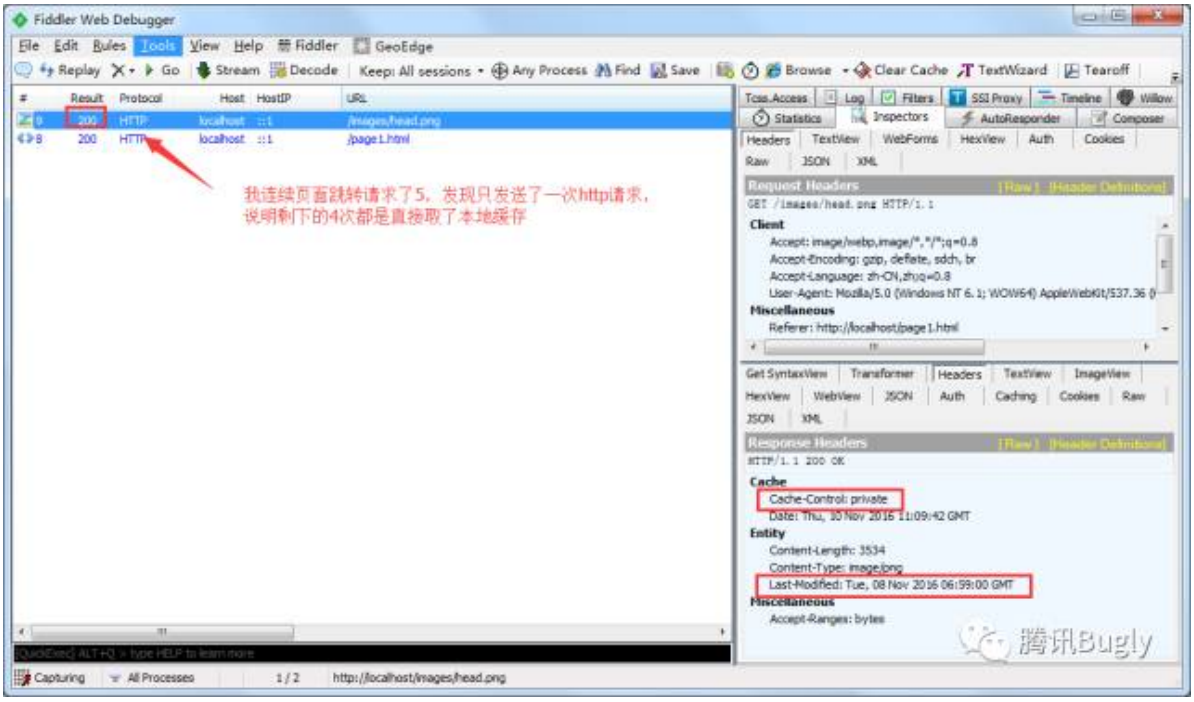
第二道题：

```
HTTP/1.1 200 OK
Cache-Control: private
Content-Type: image/png
Last-Modified: Tue, 08 Nov 2016 06:59:00 GMT
Accept-Ranges: bytes
Date: Thu, 10 Nov 2016 02:48:50 GMT
Content-Length: 3534
```

解题思路和上题一样，首先先找到缓存相关项：

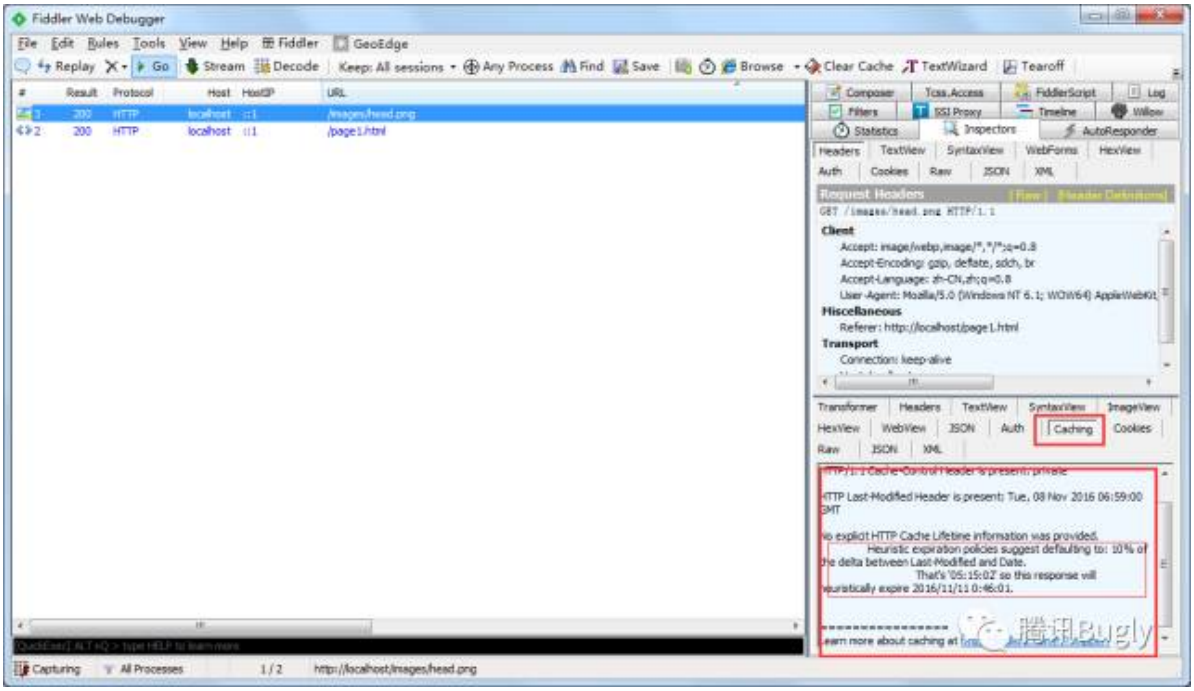
```
Cache-Control: private
Last-Modified: Tue, 08 Nov 2016 06:59:00 GMT
```

这时我们会发现根本找不到缓存过期策略项，那答案会不会和上面一样？ 一时半会也分析不出答案，那只能实际测试下了：



再看看 Chrome 浏览器下抓包：

可以看到，浏览器后续请求都直接取的本地缓存，看来的确存在某种缓存过期策略（根据我上面的缓存过期策略理论，浏览器如果直接从本地加载缓存数据，说明它相信本地缓存数据有效，那一定存在某种缓存过期判断条件）。这个问题百思不得其解，困扰了我好久，直到一次偶然的机会我在 Fiddler 响应信息面板里的 Caching 选项卡中找到了答案：



原来，在没有提供任何浏览器缓存过期策略的情况下，浏览器遵循一个启发式缓存过期策略：

根据响应头中2个时间字段 Date 和 Last-Modified 之间的时间差值，取其值的10%作为缓存时间周期。

贴一下Caching面板里的描述，英语好的同学可以精准翻译下：

```
HTTP/1.1 Cache-Control Header is present: private
HTTP Last-Modified Header is present: Tue, 08 Nov 2016 06:59:00 GMT
No explicit HTTP Cache Lifetime information was provided.
Heuristic expiration policies suggest defaulting to: 10% of the delta between Last-Modified and Date.
That's '05:15:02' so this response will heuristically expire 2016/11/11 0:46:01.
```

最终我们得到了以下完整的缓存策略三要素：

缓存策略类型	缓存策略值	结果	备注
缓存存储策略	cache-control: private	响应数据会被缓存到客户端	
缓存过期策略	Expires: 当前时间 + (date - lastModified) * 10%	根据计算公式得到一个缓存过期时间	在没有提供任何浏览器缓存过期策略的情况下，客户端计算响应头中2个时间字段Date和Last-Modified之间的时间差值(单位：秒)，取该值的10%作为缓存过期周期
缓存对比策略	Last-Modified: Tue, 08 Nov 2016 06:59:00 GMT	浏览器会携带该值去服务端比对，比对成功则返回304，服务端提示浏览器从本地加载数据，否则返回200并响应数据。	腾讯Bugly

最终结果

浏览器会根据 Date 和 Last-Modified 之间的时间差值缓存一段时间，这段时间内会直接使用本地缓存数据而不会再去请求服务器（强制请求除外），缓存过期后，会再次请求服务端，并携带上 Last-Modified 指定的时间去服务器对比并根据服务端的响应状态决定是否要从本地加载缓存数据。

总结

Http 缓存设置起来并不复杂，但却容易被轻视， 今天这篇文章结合2道题目，通过分析、解剖相关缓存头，从系统化角度对 Http 缓存机制做了一个较完整的剖析：Http 缓存机制实际上是 Http 缓存策略三个要素（纬度）相互作用的集合，所以在分析和设置 Http 报文缓存头时，只要能从中精准的分解出缓存三要素，我们就能非常准确的预判到缓存设置最终能达到的效果